CHISEL - NGÔN NGỮ XÂY DỰNG PHẦN CỨNG TRONG NGÔN NGỮ BẬC CAO SCALA HỖ TRỢ MÔ TẢ MỨC TRUYỀN THANH GHI

Lê Văn Thanh Vũ*, Trần Thị Kiều

Khoa Điện, Điện tử & CNVL, Trường Đại học Khoa học, Đại học Huế

Email: vulvt@hueuni.edu.vn

Ngày nhận bài: 17/6/2022; ngày hoàn thành phản biện: 22/6/2022; ngày duyệt đăng: 4/8/2022

TÓM TẮT

Trong bài báo này chúng tôi giới thiệu Chisel, là ngôn ngữ xây dựng phần cứng được nhúng trong ngôn ngữ lập trình bậc cao Scala. Ngôn ngữ này hướng đến cả hai nhóm lập trình: kỹ sư thiết kế phần cứng và lập trình viên phần mềm. Các kỹ sư thiết kế phần cứng tiến hành xây dựng các hệ thống điện tử số bằng các ngôn ngữ mô tả phần cứng (HDL như: VHDL, Verilog). Những người khác cần sử dụng các ngôn ngữ lập trình để tạo ra phần cứng. Chisel mang lại sự đột phá trong kỹ thuật lập trình, như với các ngôn ngữ hàm và hướng đối tượng để hình thành nên công cụ thiết kế số. Chisel không chỉ cho phép trích xuất mô tả phần cứng mức thanh ghi mà còn cho phép chúng ta viết đoạn mã tạo tạo ra phần cứng [3]. Nội dung chính của bài báo là trình bày tổng quan các thành phần của Chisel và định hướng sử dụng công cụ này trong chu trình nghiên cứu thiết kế phần cứng.

Từ khóa: Ngôn ngữ thiết kế phần cứng, Chisel, thiết kế vi mạch, công cụ hỗ trợ thiết kế.

1. MỞ ĐẦU

Lĩnh vực thiết kế phần cứng cho các hệ thống điện tử là xu hướng phát triển rất đáng được quan tâm, nhất là với các đối tượng sinh viên chuyên ngành Điện tử - Viễn thông. Hơn nữa, khi thế giới đang bị ảnh hưởng của dịch bệnh và đứt gãy chuỗi cung ứng thì sự thiếu hụt vi mạch đã và đang ảnh hưởng lớn đến rất nhiều ngành nghề sản xuất khác, như: sản xuất xe ôto, các thiết bị điện tử nghe nhìn,… Hoạt động nghiên cứu thiết kế vi mạch lại luôn cần các công cụ hỗ trợ xuyên suốt quá trình làm việc, nhưng lại khó tiếp cận vì giá thành của những công cụ này rất cao.

Chisel được phát triển từ ngôn ngữ nhúng Scala hỗ trợ tối ưu cho người dùng trong hầu hết các công đoạn nghiên cứu thiết kế từ việc mô tả thiết kế bằng mã nguồn đến khả năng triển khai mô phỏng đánh giá [1][4]. Chisel cung cấp phần lớn các thành phần cơ bản phổ biến bên trong các hệ thống số, như: đường dẫn (*wire*), thanh ghi

(*Reg*), ghép kênh (*Mux*). Các khối cơ bản này được hỗ trợ ở dạng biến hoặc hàm đã được tối ưu, người dùng chỉ cần khai báo đúng định dạng phù hợp với mục tiêu sử dụng. Ngoài ra, Chisel tích hợp khả năng viết kịnh bản đánh giá song song với quá trình mô tả mạch điện chức năng. Công cụ tích hợp hàm tạo lập cũng như trích xuất tín hiệu từ khối thiết kế một cách linh hoạt. Chisel còn cho phép người dùng sử dụng kịch bản đánh giá để tại ra tập tin "*.vcd" có thể quan sát được dạng sóng của tất cả các tín hiệu thành phần bên trong thiết kế bằng phần mềm "*Gtkwave*" [5].

Trong bài báo này tập trung giới thiệu các vấn đề cơ bản để người dùng tiếp cận nhanh công cụ thiết kế mới là Chisel theo cách trực quan nhất, đồng thời cũng trình bày bao quát các thành phần cần có để thiết kế các hệ thống số phổ biến hiện nay. Các thành phần cơ bản cấu thành nên Chisel trong hoạt động thiết kế hệ thống số sẽ được trình bày trong phần 2. Phần 3 tập trung trình bày các vấn đề liên quan đến hoạt động kiểm tra đánh giá mà Chisel cung cấp cho người sử dụng. Phần 4 là kết luận các nội dung đã trình bày trong bài báo này.

2. CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ CHISEL

Chisel về cơ bản là một ngôn ngữ lập trình và sử dụng tập các nguyên tắc cơ bản để người dùng xây dựng chương trình thực hiện các nhiệm vụ chức năng cần cho mục tiêu thiết kế. Điểm khác biệt lớn nhất của Chisel với một số ngôn ngữ lập trình thông dụng là khả năng hỗ trợ xử lý song song và tổ chức khối tuần tự hoạt động theo xung nhịp hệ thống. Một điểm đáng chú ý của Chisel với người mới sử dụng là trình biên dịch sẽ tự động chèn thêm tín hiệu xung nhịp và tín hiệu tái khởi động "*rst_n*" trong quá trình tổng hợp mã nguồn Verilog từ mã nguồn Chisel của người sử dụng. Bằng cách tạo nên mã nguồn Verilog từ Chisel, người dùng có thể sử dụng mã nguồn này để tổng hợp lên phần cứng khi sử dụng các công cụ hỗ trợ thiết kế khác nhau[5].

2.1. Các kiểu dữ liệu của Chisel

Chisel là một ngôn ngữ lập trình, các kiểu dữ liệu được dùng để mô tả kiểu giá trị được thực thi trong các câu lệnh thành phần hoặc của các đường dây và luồng tin bên trong hệ thống số. Đối với đa số các ngôn ngữ đặc tả, kiểu dữ liệu tập trung vào định dạng vector của con số nhị phân điều này giúp cho việc thể hiện giá trị cụ thể và công cụ biên dịch dễ dàng tối ưu mạch điện. Trong Chisel, một giá trị nhị phân được biểu diễn trong kiểu BITS; giá trị nguyên được thể hiện ở dạng chuỗi bit có độ dài xác định theo dạng SInt hoặc UInt. Ngoài ra, các giá trị logic sẽ được biểu diễn bằng kiểu Bool. Để thuận tiện cho hoạt động mô tả mạch mạch, Chisel bổ sung kiểu *Bundles* là tập các giá trị của các trường độc lập (tương đường với kiểu cấu trúc của các ngôn ngữ lập trình) và kiểu *Vecs* là tập các trường cùng kiểu.

Giá trị của hằng số hoặc biến thực được biểu thị qua kiểu nguyên (*interger*) của Scala hoặc dạng chuỗi thông qua khởi tạo như Hình 1. Chú ý, với các giá trị dạng chuỗi dài có thể sử dụng ký tự "_" để phân cách để dễ nhận biết nhưng sẽ được bỏ qua trong hoạt động biên dịch giá trị đó.

```
UInt(7)                 // Giá trị 4bit từ Scala
UInt("h0E")             // Giá trị 4bit hệ 16 dạng chuỗi
UInt("b01110011")       // Giá trị 8bit hệ 2 dạng chuỗi
SInt(12)                // Giá trị nguyên 4bit có dấu
SInt(-9)                // Giá trị nguyên âm 4bit
Bool(true)              // Giá trị logic từ kiểu nguyên Scala
UInt("h0A12_753B")      // Giá trị 32bit 0x0A12753B
```

Hình 1. Biểu diễn giá trị vô hướng trong Chisel.

Ở chế độ mặc định, trình biên dịch Chisel sẽ tối ưu kích thước cho các giá trị hằng và bổ sung 1bit dấu cho các giá trị có dấu. Tuy nhiên, người dùng có thể chỉ định độ dài của giá trị bằng cách bổ sung thêm giá trị độ dài ngay trong quá trình khởi tạo. Như để tạo giá trị 8bit không dấu có thể sử dụng định dạng: "*UInt("h7",8)*".

2.2. Các mạch logic tổ hợp

Mạch tổ hợp là một thành phần cơ bản luôn có trong các hệ thống số, nhưng trong các ngôn ngữ đặc tả thì mạch tổ hợp có một số hạn chế nhất định (ví dụ như ngôn ngữ Verilog sẽ không hỗ trợ việc sử dụng biến kiểu "*wire*" bên trong các khối con). Trong Chisel, mạch logic được xem là đồ hình liên kết các "*node*". Mỗi "*node*" là toán tử phần cứng, và cũng là điểm nguồn của các đường dẫn-*wire* . Để thực hiện các chức năng logic bên trong các mạch tổ hợp ta có thể sử dụng các biểu thức logic tương tự như trong các ngôn ngữ lập trình cơ bản. Ví dụ: mạch tổ hợp của 4 đầu vào {a, b, c, d} có thể thực thi chỉ với một dòng lệnh:

```
io.z = (io.a & io.b) | (~io.c ^ io.d)
```

Trong đó: io.z là tín hiệu đầu ra; các tín hiệu đầu vào là io.a, io.b, io.c & io.d.

Cú pháp sử dụng các toán tử tương đồng với Scala hoặc C++, như "&", "|", "~" tương ứng với các phép toán logic là AND, OR và NOT. Tương đồng với Verilog, chúng ta có thể đặt tên cho các biểu thức với theo kiểu đường dẫn "*wire*" theo cả hai định dạng có độ dài xác định hoặc không. Trong trường hợp không chỉ định độ dài thì trình biên dịch sẽ tự động lựa chọn độ dài tối ưu dựa theo các toán hạng trong biểu thức. Để hạn chế các trường hợp quá tải trong các thiết kế phức hợp, chúng ta có thể chia nhỏ các biểu thức bằng cách sử dụng các biến trung gian kiểu "*wire*" từng nhóm nhỏ phù hợp.

Bảng 1: Các toán tử của Chisel cho các kiểu dữ liệu[2][5]

Ví dụ	Giải thích	Độ rộng
Toán tử BIT, kiểu SInt, UInt & Bool		
val invertA = ~A	Đảo logic từng BIT	Độ rộng toán tử
val ANDconst = X & UInt("habcd_4321")	AND của X và abcd_4321	Độ rộng toán tử nhỏ nhất
val ORvar = valX | valY	OR từng bit của hai biến	Độ rộng toán tử lớn nhất
val XORvar = valB ^ valC	XOR từng bit của hai biến	Độ rộng toán tử lớn nhất
Toán tử logic bit lược giản		
val ANDvar = andR(x)	ANDvar = ANDvar ^ X	Độ rộng toán tử lớn nhất
val ORvar = orR(x)	ORvar = ORvar | X	Độ rộng toán tử lớn nhất
Phép toán so sánh		
val equVar = x===y	Phép toán so sánh bằng	Độ rộng toán tử lớn nhất
val neqVar = x !=y	Phép toán so sang khác	Độ rộng toán tử lớn nhất
Phép dịch		
val SHL = x << n	Dịch trái x *n*bit	
val SHR = x >> n	Dịch phải x *n*bit	
Phép toán nhóm bit		
val lsbit = x(0)	Tách lấy bit chỉ số 0	1bit
val Onebyte = x(15,8)	Tách lấy nhóm bit	Khoảng bit được lấy
val Catvar = Cat(var1,var2)	Hợp hai biến	Tổng hai độ rộng
Phép toán logic		
val ready = !busy	Logic đảo	Độ rộng toán tử
val condi = caseA && caseB	Logic AND	Độ rộng nhỏ nhất
val comb = stsA || stsB	Logic OR	Độ rộng lớn nhất
val Mux2 = Mux(sel,A,B)	Hợp kênh hai đầu vào	Độ rộng lớn nhất
Phép toán số học		
val sum = a + b	Phép tổng	Độ rộng lớn nhất
val diff = a – b	Phép trừ	Độ rộng lớn nhất
val prod = a * b	Phép nhân	Tổng hai độ rộng
val div = a / b	Phép chia	Độ rộng lớn nhất
val mod = a % b	Phép lấy dư	Hiệu hai độ rộng

2.3. Kiểu BUNDLE và VEC

Bundle và Vec trong ngôn ngữ lập trình được xem là một lớp đối tượng, mà trong đó cho phép người dùng thiết lập các thành phần với các kiểu khác nhau. Đây là một thành phần cơ bản được sử dụng phổ biến trong hầu hết các thiết kế hệ thống số cho các đối tượng tương đồng như các giao tiếp của khối, nhóm các đường dẫn đồng dạng. Vec thường được sử dụng như là tập các thành phần đồng dạng (như SInt, UInt hoặc Bool) và được định danh bằng chỉ số bên trong đối tượng đã được định nghĩa. Ngoài ra, Chisel còn hỗ trợ việc định nghĩa Vec là đối tượng có cấu trúc phức hợp đồng dạng, điều này không được Verilog hỗ trợ.

Bundle là một điểm rất mới trong Chisel so với các ngôn ngữ đặc tả phổ biến khác. Bundle được xây dựng hướng đến sự tối ưu hóa trong khai báo nhóm đối tượng phức hợp như: giao tiếp giữa các khối con, tập các khối con thường được tái sử dụng

trong thiết kế. Bundle là tập các thành phần phức hợp độc lập, và thậm chí có thể gồm cả kiểu Vec bên trong để trỏ đến tập các đối tượng đồng dạng.

Trong quá trình thiết kế phần cứng, hoạt động giao tiếp giữa các khối thành phần luôn được thể hiện thông qua khối cổng giao tiếp – PORT. Chisel hỗ trợ khai báo cổng bằng kiểu Bundle trỏ trực tiếp đến các thành phần bên trong gồm Input() và Output().

Sau khi kiểu của cổng được định nghĩa thì có thể sử dụng như một kiểu để sử dụng trong các khối thiết kế hoặc đặt tên cho các đường dẫn trong của thiết kế. Để linh hoạt hơn, Chisel còn hỗ trợ khả năng định nghĩa sau khi khai báo bằng các chỉ định "asInput" và "asOutput"[1].

2.4. Khối cơ bản trong Chisel và phân cấp thiết kế

Chisel vẫn giữ nguyên tắc tạo khối tương đồng với Verilog, tuy nhiên mỗi khối thiết kế trên Chisel là một lớp đối tượng, mà trong đó các thành phần của nó cũng có thể xem là đối tượng con và có đầy đủ đặc tính của đối tượng như với các ngôn ngữ lập trình khác. Một khối người dùng tự định nghĩa là một lớp với các thành phần sau

- Có tính thừa kế từ các khối cơ bản khác
- Chứa giao diện bao ngoài là hàm *IO(new Bundle {})* được đặt tên mặc định là "io"
- Các đường dẫn kết nối các kiến trúc bên trong của khối đó

Như trong Hình 2 là một khối tổng 2bit được thực hiện bằng ngôn ngữ Chisel.

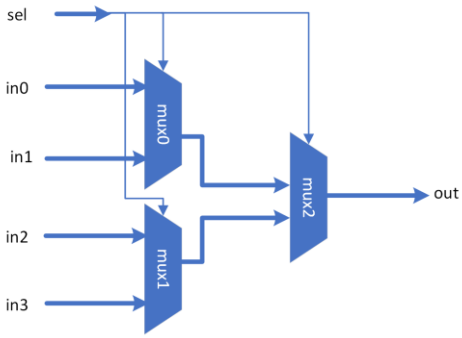
```
class   adder          extends Module {
            vai io = IO (new Bundle {
                    val a     = Input(UInt(2.W))
                    val b     = Input(UInt(2.W))
                    val c     = Output(UInt(2.W))
                    val over  = Output(Bool())
                 })
            val temp       = io.a + io.b
            io.c           :=     temp
            io.over        :=     io.a(1) & io.b(1)
    }
```

Hình 2. Cấu trúc khối thiết kế cơ bản trong Chisel

Thiết kế phần cứng luôn có tính kế thừa và tái sử dụng nên mỗi thiết kế cuối cùng thường là sự tổng hợp của nhiều thiết kế nhỏ hơn trước đó; thậm chí các thiết kế tái sử dụng lại có nhiều thiết kế con bên trong. Hình thành mô hình phân cấp thiết kế

giúp người phát triển vừa có khả năng tiết kiệm công sức khi tái sử dụng được các khối có chức năng tương đồng vừa tăng khả năng quản lý mã nguồn một cách hiệu quả hơn. Chisel hỗ trợ nguyên lý phân cấp này tương đồng với phương pháp tái sử dụng các lớp đối tượng, trong đó mỗi đối tượng là một khối thiết kế con. Hơn nữa, người dùng có thể sử dụng tham số cấu hình để thay đổi thông số của thiết kế một cách linh hoạt thông qua các cú pháp truy xuất đến các khối con. Hoạt động gọi các khối con cần được thực hiện qua hai bước: ① là gọi đối tượng thiết kế con và ② gán các đường dẫn nối giữa các khối con thông qua phép toán ":=". Một điểm đáng chú ý là trong phép gán tín hiệu nguồn phát tín hiệu ở phía bên phải của ký hiệu gán, nên khi nối đến tín hiệu đầu vào khối con cần đặt bên trái; ngược lại đối với tín hiệu ra cần đặt bên phải phép phép gán tín hiệu.

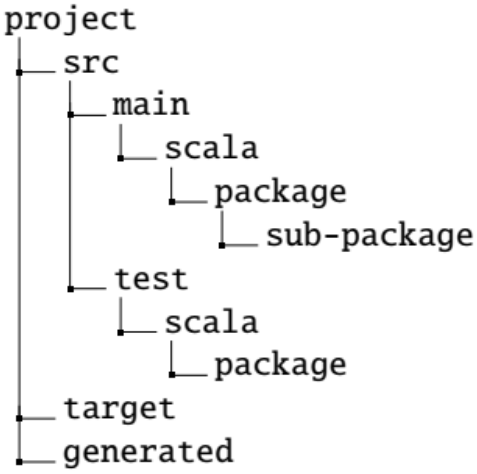
```
class   Mux4    extends Module {
    val io = IO(new Bundle{
        val in0     = Input(UInt(2.W))
        val in1     = Input(UInt(2.W))
        val in2     = Input(UInt(2.W))
        val in3     = Input(UInt(2.W))
        val sel     = Input(UInt(2.W))
        val out     = Output(UInt(2.W))
    })

        val w0      = WireInit(0.U(2.W))
        val w1      = WireInit(0.U(2.W))

    val mux0     = Module(new mux21())
        mux0.io.sel :=  io.sel(0)
        mux0.io.in0 :=  io.in0
        mux0.io.in1 :=  io.in1
        w0          :=  mux0.io.out

    val mux1     = Module(new mux21())
        mux1.io.sel :=  io.sel(0)
        mux1.io.in0 :=  io.in2
        mux1.io.in1 :=  io.in3
        w1          :=  mux1.io.out

    val mux2     = Module(new mux21())
        mux2.io.sel :=  io.sel(1)
        mux2.io.in0 :=  io.in0
        mux2.io.in1 :=  io.in1
```



Hình 3. Thiết kế phân cấp trong Chisel

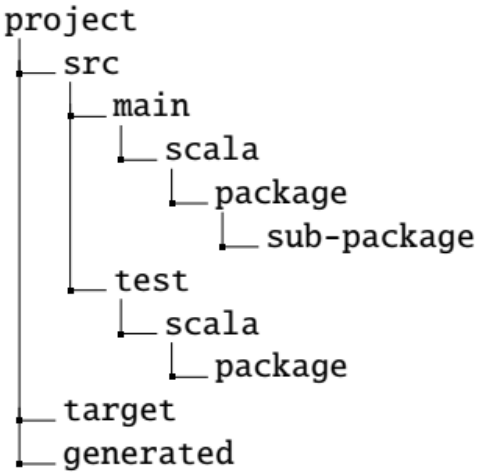
2.5. Xây dựng dự án thiết kế

Người dùng Chisel cần xây dựng một dự án thiết kế trên một thư mục tổng để quản lý được tất cả các tập tin và nội dung liên quan thông qua cây thư mục cơ bản như trong



Hình ***4***. Trong đó, toàn bộ mã nguồn của dự án là lưu trong các tập tin có đuôi "*.scala" được bố trí trong các thư mục bên trong thư mục "***src***"; kể các nhóm mã

nguồn cho hoạt động kiểm tra đánh giá được lưu trong nhánh "***test***" của thư mục "***src***". Kết quả biên dịch sẽ tạo ra các tập tin mã nguồn Verilog và tập tin dạng song trong thư mục "***generated***".



Hình 4. Cấu trúc cây thư mục cơ bản của một dự án trong Chisel (Schoeberl, 2019).

Chisel được xây dựng dựa trên SBT và ngôn ngữ Scala nên cần có nền tảng Java hỗ trợ [1]. Quá trình cài đặt hệ thống cần môi trường Javal hỗ trợ, sau đó cài thêm trình biên dịch của ngôn ngữ Scala phù hợp. Sau khi cài các công cụ hoàn chỉnh, người dùng có thể đăng nhập GitHub để tải về một số dự án cơ bản của Chisel trong đó đã có hỗ trợ các đoạn "*script*" khai báo môi trường có sẵn để thực thiện quá trình cấu hình công cụ, biên dịch và khởi tạo kết quả một cách dễ dàng.

3. QUÁ TRÌNH KIỂM TRA ĐÁNH GIÁ THIẾT KẾ TRONG CHISEL

Hoạt động nghiên cứu thiết kế phần cứng điện tử luôn đi kèm với hoạt động mô phỏng đánh giá, Hoạt động này vừa có ý nghĩa kiểm tra cú pháp vừa kiểm tra từng bước hoạt động của thiết kế tạo ra. Chisel hỗ trợ mô phỏng thiết kế đánh giá một cách trực quan và đơn giản thông qua việc cung cấp các hàm khởi tạo dữ liệu đầu vào cho thiết kế, hàm đánh giá so sánh kết quả đầu ra. Người dùng Chisel có thể linh hoạt lựa chọn giải pháp đánh giá dựa vào quan sát dạng sóng (*waveform*) hoặc sử dụng ngay các hàm đánh giá được cung cấp để xác định được kết quả mô hình thiết kế ngay tại quá trình biên dịch thiết kế.

Về cơ bản, Chisel vẫn hỗ trợ thiết kế đánh giá tương đương một khối ở mức cao, trong đó chứa khối thiết kế cần đánh giá – DUT (Design Under Test), khối logic đánh giá và đối tượng đánh giá chứa các hàm để khởi động và đánh giá. Đặc điểm quan trọng nhất là của thiết kế đánh giá là khối thiết kế sẽ không có giao diện vào/ra; và cần chỉ định rõ đối tượng cần đánh giá – DUT. Trong Chisel phiên bản 3 hỗ trợ chức năng đánh giá trong đóng gói "*iotesters*" nhiều lớp hỗ trợ cho đa dạng các kỹ thuật

đánh giá phổ biến hiện nay. Trong bài báo này giới thiệu lớp con được sử dụng phổ biến và hiệu quả là "*PeekPokeTester*" với các chức năng hỗ trợ trực quan cho người thiết kế gồm:

1. Hàm khởi tạo đối tượng đánh giá:
   - ✓ newPeekPokeTester(dut: T, base: Int = 16, logFile: Option[File] = None)
2. Hàm tạo giá trị cổng vào:
   - ✓ def poke(signal: Aggregate, value: IndexedSeq[BigInt]): Unit
   - ✓ def poke(signal: Bundle, map: Map[String, BigInt]): Unit
   - ✓ def poke[T <: Element](signal: T, value: Long)(implicit arg0: Pokeable[T]): Unit
   - ✓ def poke[T <: Element](signal: T, value: Int)(implicit arg0: Pokeable[T]): Unit
   - ✓ def poke[T <: Element](signal: T, value: BigInt)(implicit arg0: Pokeable[T]): Unit
   - ✓ def poke(path: String, value: Long): Unit
   - ✓ def poke(path: String, value: Int): Unit
   - ✓ def poke(path: String, value: BigInt): Unit
3. Hàm truy xuất giá trị cổng ra
   - ✓ def peek(signal: Bundle): LinkedHashMap[String, BigInt]
   - ✓ def peek(signal: Aggregate): Seq[BigInt]
   - ✓ def peek[T <: Element](signal: T)(implicit arg0: Pokeable[T]): BigInt
   - ✓ def peek(path: String): BigInt
4. Hàm so sánh kết quả
   - ✓ def expect(signal: Bundle, expected: Map[String, BigInt]): Boolean
   - ✓ def expect(signal: Aggregate, expected: IndexedSeq[BigInt]): Boolean
   - ✓ def expect [T<: Element] (signal: T, expected: Int, msg:⇒ String)
     (implicit arg0: Pokeable[T]): Boolean
   - ✓ def expect[T <: Element](signal: T, expected: BigInt, msg:
     ⇒ String = "")(implicit arg0: Pokeable[T]): Boolean
   - ✓ def expect(good: Boolean, msg: ⇒ String): Boolean

4. KẾT LUẬN

Chisel vừa là ngôn ngữ hỗ trợ thiết kế phần cứng khi công cụ này cung cấp đầu đủ các thành phần để người dùng có thể xây dựng một hệ thống tích hợp hoàn chỉnh. Đồng thời, Chisel còn cung cấp các thành phần cơ bản hỗ trợ người dùng thực hiện nhanh và tối ưu hơn trong việc mô tả chi tiết như "Mux, Reg, RegInit, …".

Quan trọng hơn là Chisel cung cấp khả năng mô phỏng đánh giá và kiểm thử thiết kế một cách linh hoạt và trực quan khi vừa có thể cung cấp tập tin "*.vcd" để kết hợp với ứng dụng "gtkwave" để quan sát dạng sóng vừa có thể sử dụng trực tiếp đối tượng so sánh ngay trong "*iotesters*".

Qua quá trình nghiên cứu tìm hiểu Chisel và bước đầu ứng dụng vào quá trình nghiên cứu thiết kế vi mạch đã giúp cho chúng tôi tiết kiệm thời gian cả trong quá trình viết mã nguồn và mô phỏng đánh giá. Mặc dù Chisel không trực tiếp cung cấp môi trường tích hợp hỗ trợ thiết kế, nhưng người dùng có thể sử dụng đa dạng các môi trường soạn thảo phổ biến như: Vim, Notepad, Eclipse, …

TÀI LIỆU THAM KHẢO

[1] Jonathan Bachrach, H. V. (2012). Chisel: Constructing Hardware in a Scala Embedded Language. DAC Design Automation Conference.

[2] Jonathan Bachrach, K. A. (2014). Chisel 2.2 Tutorial. Retrieved from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.681.6761&rep=rep1&type=pdf.

[3] Le Van Thanh Vu, T. H. (2021). Đề xuất kiến trúc và đánh giá thử nghiệm khối mã hóa không dùng sbox cho hoạt động truyền thông của hệ thống IoT. Hue: Tạp chí Khoa học công nghệ - Đại học Khoa học Huế.

[4] Liu, X. a. (2013). Chisel-Q: Designing quantum circuits with a scala embedded language. 2013 IEEE 31st International Conference on Computer Design (ICCD), (pp. 427-434).

[5] Schoeberl, M. (2019). Digital Design with Chisel. USA: Kindle Direct Publishing.

CHISEL – HARDWARE CONSTRUCTION LANGUAGE IN HIGH-LEVEL PROGRAMMING LANGUAGE SCALA SUPPORT RTL DESCRIPTION

Le Van Thanh Vu*, Tran Thi Kieu

Faculty of Electronics, Electrical Engineering and Material Technology

University of Sciences, Hue University

Email: vulvt@hueuni.edu.vn

ABSTRACT

In this paper, we introduce Chisel - the hardware construction language embedded in the high-level programming language Scala. This language targets two groups: hardware designers and software programmers. The hardware designers build digital systems by using the hardware description languages (VHDL, Verilog). The others use a programming language for generating hardware. Chisel brings advances in software engineering, such as object-orientated and functional language into digital design. Chisel does not only allow you to express hardware at the register-transfer level but also allows you to write hardware generators. This paper aims to present an overview of the Chisel's components and the orientation to use this tool in the hardware design works.

Keywords: HDL, Chisel, Scala, IC design, CAD tool.

Lê Văn Thanh Vũ sinh ngày 20/05/1977 tại TP Huế. Ông nhận bằng cử nhân đại học ngành Vật lý tại Trường Đại học Khoa học, Đại học Huế. Năm 2004, ông nhận bằng thạc sỹ ngành Điện tử - Viễn thông tại Khoa Công nghệ thuộc Đại học Quốc gia Hà Nội. Năm 2017, ông nhận bằng tiến sĩ tại Trường ĐH Công nghệ - ĐHQG Hà Nội. Hiện đang là giảng viên Trường Đại học Khoa học – Đại học Huế.

Lĩnh vực nghiên cứu: Thiết kế vi mạch, hệ thống nhúng – IoT.

Trần Thị Kiều sinh ngày 29/02/1992. Năm 2015, bà nhận bằng kỹ sư ngành Điện tử - Viễn thông tại khoa Điện tử - Viễn thông thuộc trường Đại học Khoa học - Đại học Huế. Hiện nay, bà đang học cao học tại Khoa Công nghệ thông tin thuộc trường Đại học Khoa học – Đại học Huế.

Lĩnh vực nghiên cứu: Khoa học máy tính.